Naval Research Laboratory

Stennis Space Center, MS 39529-5004

NRL/MR/7440--14-9497

Customized Architecture for Complex Routing Analysis: Case Study for the Convey Hybrid-Core Computer

Chris J. Michael
Elias Z. Ioup
David W. Dobson

Geospatial Sciences and Technology Branch
Marine Geosciences Division

February 18, 2014

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. **PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.**

1. REPORT DATE *(DD-MM-YYYY)* 18-02-2014	**2. REPORT TYPE** Memorandum Report	**3. DATES COVERED** *(From - To)* 01-01-2013 – 22-07-2013

4. TITLE AND SUBTITLE Customized Architecture for Complex Routing Analysis: Case Study for the Convey Hybrid-Core Computer	**5a. CONTRACT NUMBER**
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER 74-9352-A3
6. AUTHOR(S) Chris J. Michael, Elias Z. Ioup, and David W. Dobson	**5d. PROJECT NUMBER**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Research Laboratory, Code 7442 Marine Geosciences Division Stennis Space Center, MS 39529-5004	**8. PERFORMING ORGANIZATION REPORT NUMBER** NRL/MR/7440--14-9497
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Geospatial-Intelligence Agency 7500 GEOINT Drive Springfield, Virginia 22150	**10. SPONSOR / MONITOR'S ACRONYM(S)** NGA
	11. SPONSOR / MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

For conducting complex routing analysis, FPGAs could prove a worthy candidate over conventional commodity processors if the cost-to-performance ratio is significant. This report presents the background, experiments, and results of a scaling study that compares the FPGA-based Convey "Hybrid Core" architecture to a modern conventional high-performance node of equal form factor in performing all-pairs shortest paths on networks of streets. Results show that the Convey system is able to yield a result over 5 times faster for a graph of 3 million nodes. Moreover, there is still significant room for further optimization, while the conventional system implementation is optimized to the point of diminishing returns.

15. SUBJECT TERMS

Computer architecture	Navigation
Routing	High performance computing
Graph processing	Scaling

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** Chris J. Michael
a. REPORT Unclassified Unlimited	**b. ABSTRACT** Unclassified Unlimited	**c. THIS PAGE** Unclassified Unlimited	Unclassified Unlimited	12	**19b. TELEPHONE NUMBER** *(include area code)* (228) 688-4955

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39.18

Customized Architecture for Complex Routing Analysis: Case Study for the Convey Hybrid-Core Computer

1 Abstract

Because conventional commodity processors, though inexpensive, do not implement the best architecture to exploit complex routing analyses, customized architecture via FPGAs (Field Programmable Gate Arrays) could prove a worthy candidate if the cost-to-performance ratio is significant. In this report, the Naval Research Laboratory (NRL) will present the background, experiments, and results of a scaling study that compares the FPGA-based Convey "Hybrid Core" architecture to a modern conventional high-performance node of equal form factor in performing *all-pairs shortest paths* (APSP) on graphs representing routing networks of streets. The results will help determine cost/performance return for graph algorithms operating on large routing networks using the Convey architecture.

Results show that a highly optimized APSP implementation written in C on a conventional system does not scale well for graphs larger than 1 million nodes, while a moderately optimized Convey implementation scales well for these and larger graphs, presumably scaling up to 10 million nodes. The Convey APSP implementation is able to yield a result over 5 times faster for the largest tested input graph of 3 million nodes. Moreover, there is still significant room for further optimization on the Convey implementation while the conventional system implementation has already been optimized to the point of diminishing returns.

2 Experimental Framework

2.1 Hardware Specification

We chose a conventional "baseline" system optimized for memory bandwidth and a fairly recent mid-range Convey system. The baseline system is housed at NRL-Stennis and the Convey system is housed at Convey Computer in Richardson, TX.

2.1.1 Baseline

Table 1: Baseline System Specs

Node		
Model		Dell PowerEdge
Rack Size		2U
Number of Sockets		4
System Bus Interconnect		HyperTransport
Memory	Type	DDR3 1600MT/s
	Capacity	32 DIMM X 16GB = 512 GB
Socket (4 total in system)		
Number of Processors		2
CPU Interconnect		HyperTransport
Processor (8 total in system)		

Manuscript approved October 7, 2013.

Model		AMD Opteron 6276
Number of Cores		8 (64 total in system)
Clock		2.3GHz
Cache	Line Size	64B (16 words)
	L1 I-Cache Size	256KB (64KB/2 cores)
	L1 D-Cache Size	128KB (16KB/core)
	L2 D-Cache Size	8MB (2MB/2 cores)
	L3 D-Cache	8MB shared*

2.1.2 Convey Hybrid Core

Table 2: Convey System Specs

Node		
Model		Convey HC-2ex
Rack Size		2U
Number of Sockets		6
System Bus Interconnect		QPI
Host Memory	Type	DDR3 1600MT/s
	Capacity	192 GB
Coproc Memory	Type	Scatter-Gather DIMM
	Capacity	64 GB
Host Processor (2 total in system)		
Model		Intel Xeon X5670
Number of Cores		6
Clock		2.93GHz
Coprocessor (4 total in system)		
Model		Xilinx Virtex-6 LX760

2.2 Benchmark Application

A benchmark application has been developed both on the baseline system and on the Convey system. Each benchmark performs APSP and has the same input and output. The internal processing of each benchmark differs due to the distinct architectural differences between the baseline system and the Convey system.

2.2.1 Input

The input graphs used for the study are all directed weighted graphs extracted from NAVTEQ data. A Node in the graph either represents a street intersection or a dead-end. Edges represent adjacent connectivity between two neighboring intersections. The edge weight represents the distance between the two intersections. Nodes in the final routing graph correspond to the REF_IN_ID and NREF_IN_ID unique intersection IDs in the attributes of each feature in the NAVTEQ Streets layer. The value of the DIR_TRAVEL attribute determines whether the connection will be bi-directional or one way. Note that we do not retain or exploit any geospatial information such as coordinates. Also note that the NAVTEQ Z-Levels layer is essential for constructing these routing networks correctly.

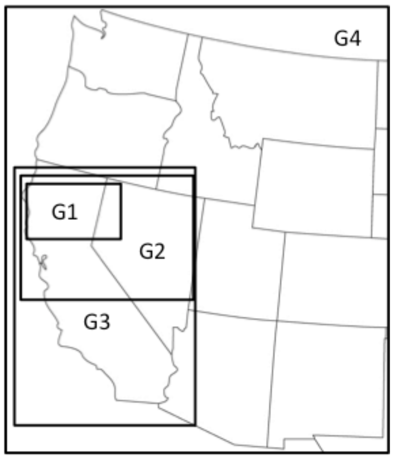

Figure 1: Routing Network Regions

The graph sizes of the data sets were chosen to be manageable while providing a good sense of performance scaling. Each set is taken from the western part of the US. Figure 1 shows the bounding boxes of each input graph. Each graph is stored in a text file, each line of which corresponds to an edge of the graph. Each line contains the source ID, destination ID, and an edge weight representing the traveling distance between the corresponding intersections. Each dataset roughly increases by an order of magnitude for N^2, where N is the number of vertices in the graph.

Some preprocessing is performed on the input to optimize execution without changing results. Node IDs are reassigned (from NAVTEQ intersection IDs) as contiguous numbers starting from zero, which allows for better performance and memory contiguity. This transformation is reversible with minimal bookkeeping. Additionally, duplicate edges, which exist profusely in the NAVTEQ data set, are removed.

2.2.2 C Implementation (Conventional Multicore)

In early experimentation, it was found that static compressed adjacency list representations such as *compressed sparse row* (CSR) improved performance significantly for APSP on graphs representing routing networks. Therefore, the input graph is implemented using CSR for the purposes of this study. This implementation has a much greater potential to exploit spatial and temporal locality due to its compact and contiguous memory footprint. For graphs over 10,000 nodes, we have found that implementing a CSR rather than an adjacency list can speed execution of APSP over 10 times.

Dijkstra's algorithm, the chosen algorithm for the study, requires a priority queue to keep track of intermediate path distances. For our implementation, we chose to use a *referenced binary heap* like the one shown in the Figure 2.

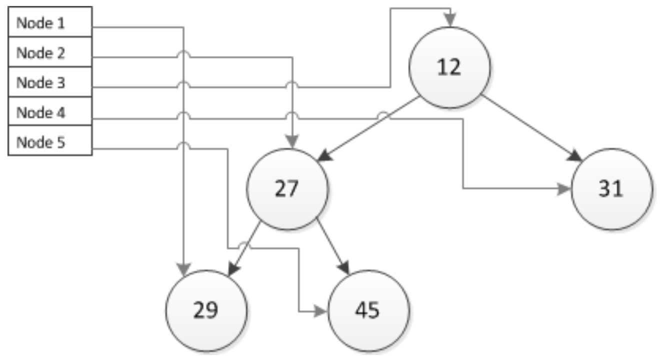

Figure 2: Referenced Binary Heap

Building the heap requires O(N log N) time. An enqueue or dequeue operation occurs in O(log N) time, which is optimal among other priority queue implementations. Dijkstra's algorithm requires updates to distances in the queue referenced by the node IDs. Due to the reference look-up table, the queue can be updated in constant time. Having this table does not impart a detrimental memory overhead for the given input sets and can speed up processing by one order of magnitude for graphs over 10,000 nodes.

The interface for this implementation is as follows.

```
    int dijkstra(graph, src, dist, pred)
```

Graph is a pointer to the CSR representation of the input graph, *src* is the ID of the source node, the return value *dist* is a pointer to the N-sized list of calculated distances, and the return value *pred* is a pointer to the N-sized list of shortest-path predecessors (all full shortest paths can be constructed by traversing the predecessor's list). The method's return value is the average single-source shortest path length for the given source node.

One instance of *dijsktra* is run as a separate OpenMP thread for each node in the graph. A single instance of the input graph is shared among all threads, and each thread will allocate both its N-sized result lists as well as its N-sized priority queue. All threads reduce to a sum accumulator so that the average APSP length for the graph may be calculated. Once the result is accumulated, *dist* and *pred* are deallocated.

2.2.3 Hybrid Threading Implementation (Convey HC2-ex)

The Convey *Hybrid Threading* (HT) toolkit is a development environment that enables Convey *personalities* to be developed using C++. The toolkit allows for easy integration between the Intel general-purpose host processors and the Convey FPGA coprocessors by providing one programming environment for implementation on both processor architectures.

The HT benchmark also uses a CSR to represent the graph, and all other input and output data structures of the HT benchmark match those of the C benchmark implementation. The interface is the same as that of *dijkstra* described above. However, due to design time constraints and the specialized Convey architecture, we chose to implement a modified version of the Bellman-Ford (BF) APSP algorithm. Though the BF algorithm requires significantly more edge traversals, it does not require a priority queue. Additionally, it has the potential to use memory bandwidth more effectively.

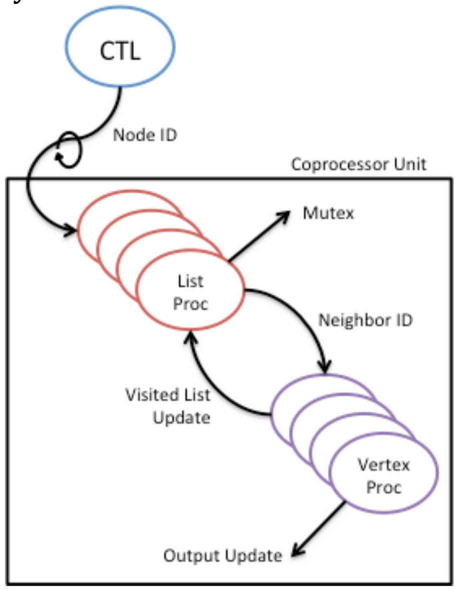

Figure 3: Diagram of HT APSP Implementation

In the HT implementation, the conventional host processor must first copy the input graph to the coprocessor's memory. Once this is done, the application instantiates a thread manager (shown as *CTL* in Figure 3) to begin issuing threads to *units*, or FPGA coprocessor cores. One thread will be issued per source node in the input graph, and each unit has memory allocated for *dist*, *pred,* and a working list for BF. Each unit contains multiple *list processors* that dispatch vertices in the working list. When a vertex is dispatched, it spawns a thread for each outgoing edge onto a *vertex processor* that is responsible for calculating new distances and updating the result. Once a unit is done executing its BF instance, it copies its full result to the host.

The heart of the HT implementation is the vertex processor. The HC-2ex can support a very large amount of vertex processor threads to be in flight at any given time. The personality was designed this way in order to maximize memory throughput. However, this causes a semaphore in the algorithm: Two threads running on vertex processors may read and update path lengths to the same vertex incorrectly. Because of this, we have implemented a *mutex* (mutual exclusion) that allows a thread to lock a node before modifying it exclusively. Because we are dealing with sparse graphs where the average in-degree is 2, the overhead involved with the mutex is trivial.

3 Results

Table 3: Results Baseline vs. Convey

	Nodes	Edges	CSR Size (MB)	Wall Time		Speedup (Convey)
				Baseline	Convey	
G1	124543	297966	3	149	134	1.11
G2	509099	1241122	12	3657	2138	1.71
G3	1253148	3047625	30	76785	24519	3.13
G4	3636185	9138912	88	718803	139628	5.15

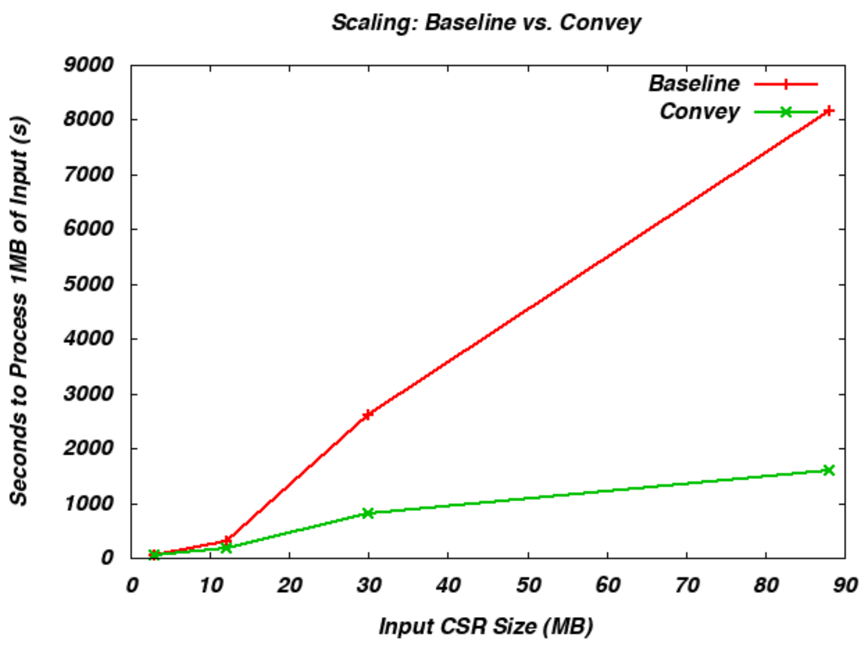

Figure 4: Results for Scaling

See Table 3 and Figure 4 above for the scalability results. The baseline system ran Dijkstra's algorithm benchmark with 64 threads. The Convey system ran the parallel BF algorithm benchmark with 64 units. Each unit has one list processor and 64 vertex processors.

The performance of the baseline system begins to suffer dramatically once the input data graph size exceeds 12MB, or 500,000 nodes. This is mainly due to poor cache performance, since both spatial locality and temporal locality diminish as the input graph reaches this size. After this point, there is a significant constant-factor overhead due to the memory inefficiency that causes the algorithm to scale poorly.

Conversely, the Convey implementation scales better as the input size gets larger. Larger input sizes help to better saturate the outstanding loads and therefore the memory throughput of the system. In the largest graph we examined, the Convey system outperformed the baseline system by over five times. Projecting the results towards the target graph of 10 million nodes (300MB input size), the Convey system is expected to maintain 5.2 times speedup, assuming it can accommodate the larger output size. For this graph size, the baseline system is projected to complete the run in 99 days, while it will take the Convey system 19 days.

4 Conclusion

It has been shown that for large routing networks on the order of millions of nodes, the Convey system can perform over 5 times better than a conventional multicore system of similar form factor. Moreover, the APSP algorithm used for the conventional system is a highly optimized and targeted algorithm, while the Convey system implementation has plenty of room for optimization of memory throughput. With optimizations to the algorithm and memory performance, we feel that a refined and optimized Convey HT implementation may reach an order of magnitude speedup within 6-months of development. Though more optimizations may be applied to the conventional implementation, the wasted memory bandwidth due to random-access memory is the biggest bottleneck and any significant performance gain is not to be expected.

Appendices

A.1 Background

In order to better understand why APSP graph algorithms perform better on Convey's architecture, we briefly present some background in graph algorithms and computer architecture relevant to this study. For further reading, please refer to the final *Further Reading* section of this appendix.

A.1.1 Graph Algorithms

There are two common ways to represent an edge-weighted directed graph in a data structure. An *adjacency list* keeps a record for every node in the graph. Each record contains a list of nodes adjacent to the corresponding node, and each element in the list has a value representing the corresponding edge weight. The adjacency list requires O(N+E) entries, where N is the number of nodes in the graph and E the number of edges. An *adjacency matrix* contains an entry for each possible edge in the matrix. The adjacency matrix consists of O(N^2) entries for edge weights. Though this storage requirement can be overwhelming for large graphs, only a very small percentage of the entries are used for a sparse graph. Therefore, compressed adjacency matrix representations such as *compressed sparse row* (CSR) can greatly reduce the size of sparse graphs, only requiring size O(N+E). In implementation, CSR representations are typically smaller than adjacency list representations because CSR does not require the storage of pointers; however, CSR representations are, for the most part, immutable.

The Bellman-Ford algorithm computes the single-source shortest paths for a directed weighted graph. The algorithm *relaxes*, or traverses through, each edge in the graph up to N times. The result contains the shortest paths and weights of these paths from a given source node to all other reachable nodes in the graph. The Bellman-Ford algorithm runs in O(NE) time.

Dijkstra's algorithm computes the single-source shortest paths for a directed weighted graph containing no negative-weighted cycles. The algorithm relaxes edges similarly to breadth-first search, and each edge is only relaxed once. The algorithm requires a priority queue to keep track of intermediate shortest-path lengths. When the priority queue is implemented as a binary heap, Dijkstra's algorithm can run in O(N log N) time. The result matches that of the Bellman-Ford algorithm.

A.1.2 Computer Architecture

Modern processors for high-performance computing exhibit a hierarchical memory layout in order to mitigate the high latency associated with requesting off-CPU memory. This hierarchy consists of multiple levels of caches that reside between physical memory and the CPU. Though these caches reflect a relatively small subset of off-CPU memory, they provide access to this memory up to 100 times faster. Caches greatly enhance the performance of algorithms that exhibit *temporal locality* and *spatial locality*. Temporal locality is the behavior of accessing a memory location multiple times within a short duration. Spatial locality is the behavior of accessing separate memory locations that are nearby one another.

Processors for high-performance computing almost always exhibit multiple CPUs, commonly referred to as cores. Each CPU has the ability to execute its own separate program, but it must share memory space with other CPUs. In order to facilitate communication across multiple CPUs, a parallel programming paradigm is essential. OpenMP is a widely utilized and robust API that efficiently implements such a paradigm. Applications using OpenMP conduct thread management to assure that all CPUs are busy running threads.

Field Programmable Gate Arrays (FPGAs) are packaged integrated circuits that can be reconfigured using a hardware description language such as Verilog. Current state-of-the-art FPGAs can handle enough programmable logic to leverage special-purpose computing that can compete or outperform conventional processors. This is especially true for problems that don't require either a large arithmetic capacity or complex control.

The Convey HC-2ex "Hybrid Core" computer is a high performance computing architecture that consists of a conventional host processor paired with a completely reconfigurable Field-Programmable Gate Array (FPGA) coprocessor. Though the coprocessor has a separate physical memory space, it shares the virtual memory address space with that of the host processor so that a parallel programming paradigm may be effectively implemented. The coprocessor does not implement a cache out of the box and has a customized memory interface and interconnect to facilitate high bandwidth and low latency.

The Convey coprocessor implements a basic instruction set architecture to facilitate transactions with the host processor. The Convey Personality Development Kit (PDK), a Verilog-based design environment, is used to implement a custom-designed computer architecture, or *personality*, to consume these instructions and special custom instructions described by the designer. The PDK offers full access to the special coprocessor memory interconnect interface, but it is up to the designer to make effective use of it. To ease in the design of personalities, Convey develops *Hybrid Threading* (HT) toolkit that allows for the generation of pipelined and threaded personalities from a software (C/C++) interface.

A.2 Performance of Oracle Spatial

Oracle Spatial is a proprietary product developed as a solution for supporting and managing geospatial data. It is a separately licensed component of the Oracle Database. The extensions provided by the component optimize storage and querry support for abstract graphs and spatial networks as well as geometry objects including points, lines, and polygons. Oracle Spatial also provides a Java API for accessing and performing operations on the stored data.

There are several methods provided in the Oracle Spatial API for calculating single-source shortest paths to all connected nodes. The most efficient method for calculating this on a relatively large and sparse abstract graph is the *ShortestPaths* function in the Java API. Oracle Spatial has the ability to optimize the operation for a spatial network, but this functionality is not covered in this report.

Table 4: Oracle Spatial Performance

10, Hawaii			7, Northeast US		
Threads	Native(s)	DB(s)	Threads	Native(s)	DB
1	231	19744	1	328024	N/A
8	41	2968	8	56556	N/A
16	22	1434	16	38206	N/A
32	11	734	32	17922	N/A
64	8	513	64	12350	N/A

Results for the total execution time required for completion of both the C APSP benchmark *Native* and the Oracle Spatial benchmark *DB* are shown in Table 4 for NAVTEQ Regions 10 (Hawaii, 41k nodes) and 7 (NE US, 1M nodes). (Note that this study was conducted with the internal NAVTEQ partitioning which limited graphs to be no more than 500k nodes. Z-Levels are needed to calculate the true routing graphs.) These benchmarks were both run on the conventional system described in the main section of this study. Both native and DB scale well for the given input, but the native benchmark is consistently over 60-times faster than DB. This is mainly due to the overwhelming amount of memory required for Oracle Spatial. The smallest run for DB on region 7 would take 3 days to complete, so these results were omitted due to lack of time and allocation on the system. This overwhelming increase in completion time is mainly due to the multiple-orders-of-magnitude increase in the amount of memory DB requires. See Table 5 for memory usage statistics.

Table 5: Oracle Spatial Memory Usage

Hawaii		
Threads	Native(MB)	DB(MB)
1	1	1600
8	8	3169
16	16	5430
32	31	11913
64	62	12321